# 

# Department of Computing

**CS 250: Data Structures and Algorithms**

**Class: BSCS-9AB**

**Lab 07: Asymptotic Complexity of Algorithms**

**CLO1: Understand the fundamentals of data structures and algorithms**

**Date: November 23, 2020**

**Time: 10:00 am -1:00pm, 2:00pm – 5:00pm**

# Instructor: Dr. Yasir Faheem

# Lab 7: Asymptotic Complexity Analysis

**Introduction**

This lab is based on the analysis of different algorithms.

**Objectives**

Objective of this lab is to make students analyze different algorithms and their asymptotic complexities.

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++

**Tasks**

**Task 1:**

You have already implemented a function that prints all elements of a list of size n, where n>=0. What is the Big-O complexity of that operation?

**CODE:**

**void printLinkList()**

{

//if list is empty,method will return

if (isEmpty())

{

cout << "nothing to show u";

return;

}

temp = start;

//condition valid tills the last of list

while (temp != last->next)

{

cout << temp->data;

//displays the element

temp = temp->next;

// temp pointer is incremented

}

cout << endl;

}

**Explaination**

While loop iterates until the whole list is accessed. So it iterates equal to number of elements in link list. The statements outside the loop takes constant time so that will not included while calculating Big O. There are two statement in while loop.so its complexity could be 2n but constant is not included in big O because no matter how much large is constant it could not exceed n\*n so it will be ignored. This algorithm has Big-O complexity is **n.**

**Big O complexity:** Big-O complexity is **n.**

**Task 2:**

In assignment 1, you were asked to implement a function that prints all elements of a singly linked list in the **reverse order.** Your task is to answer the following questions:

1. Suppose the elements of a singly linked list are printed using an iterative approach with the help of two nested loops. What is the Big-O time complexity of printing n values in the reverse order? What is the Big-O **space complexity**?

**Time Complexity**

For nested loops to print the elements in reverse order

Time complexity is n+(n-1)+(n-2)+…….1=n(n+1)/2

**O(n^2)**

1. Suppose the elements are printed using a recursive function given below. What is the Big-O time complexity of this function? What is the Big-O space complexity? Hint: stack, function calls!

Void RecursivePrint( node \*temp){

If (temp!=NULL){

RecursivePrint(temp->next);

cout<<temp->data;

}// end of if.

}

**Time Complexity** is **O(n)**

**Task 3:**

Suppose you have a **singly linked list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function? What is its best-case time complexity?

For this function, a temp will be pointed to start A count variable initialized to 0 in start of method. A while loop to iterate from start to last of link list and count is incremented each time and compared with the value of position passed as a parameter of function. If count and pos are equal it will return with value at that position

**Best Case:** It is the case when value is found at start of list. Only for one iteration in loop. So its complexity will be constant. As we enter the loop incremented the count compare with position, found equal and value will be returned.

**Best case complexity equals one.**

**Worst Case**: It is case when position passed is not in the last or at the last element of list. Loop iterates over the whole list, incrementing count compares with the position and pointing the pointer to its next. When the whole loop is iterated the function will return.

**Time Complexity=**2+3n.

So Big (O) will be equal to n.

**Task 4:**

Suppose you have an **array-based list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function? What is its best-case time complexity?

In Array based implementation of list we can directly access any index so no loop is required. As soon as function is called, passed parameter position will be given as index of array and corresponding value will be returned. So a constant time is required to do this.

**Big(O) is 1.**

Its best and worst case complexity both are equal and constant.

**Task 5:**

What is best-case and worst-case time complexity to destroy a linked list of size n?

void DestroyList()

{

temp = start;

//From start to last deletes all elements in list

while (temp != NULL)

{

start = start->next;

delete temp;

temp = start;

}

}

**Best Case Scenario:**

It is the case when list is empty. So only one statement will be executed .It has constant complexity so for best case time complexity will be 1.

**Worst Case Scenario:**

It is the other scenario if the list contains n number of element for n>0.So this loop will iterate over n times. Worst Case complexity would be equal to 1+3n. **Big(O) will be n.**

**Task 6:**

What is best-case and worst-case time complexity to destroy an array-based list of size n?

**Answer**

For destroying an array based list we only have to do one step to make the length equals to 0

Only one statement to be executed for destroying array based list.

As for any number of elements, no matter how much elements are in list.

So its best and worst case complexity will be equal and that’s constant **so big (O) will be 1.**

**Task 7:**

Your task is to reverse the order of all n elements of a singly linked list using stack. Declare a stack of pointers to class node (It should store the address to an object of class node). Travers the linked list and push the address of every node onto a stack. Pop the elements and update the links by reversing order of nodes in a list. Update the start and last pointers. What is the best case and Big-O time complexities to reverse a linked list using this approach? What is its Big-O space complexity?

#include <stdio.h>

#include <iostream>

using namespace std;

class LinkList;

class Stack;

class LinkListNode;

LinkListNode\* start;

LinkListNode\* last;

class Stack {

public:

LinkListNode\* Array[10];

int top = -1;

int i = 0;

LinkList\* temp;

void Push(LinkListNode\* node)

{

Array[i] = node;

top++;

i++;

}

LinkListNode\* pop()

{

LinkListNode\* temp = Array[top];

top--;

i--;

return temp;

}

};

class LinkListNode

{public:

int data;

LinkListNode\* next;

};

class LinkList

{public:

LinkListNode\* AddNode(int value)

{

LinkListNode\* newnode = new LinkListNode();

if (start== NULL)

{

start = newnode;

last = newnode;

}

else

{

last->next = newnode;

last = newnode;

}

last->data = value;

return newnode;

}

void Revesing()

{

Stack\* s = new Stack();

LinkListNode\* temp = start;

while (temp!= NULL)

{

s->Push(temp);

temp = temp->next;

}

int count = 0;

while (s->top != -1)

{

if (count == 0)

{

temp = s->pop();

start = temp;

count++;

}

else

{

temp->next = s->pop();

temp = temp->next;

}

if (s->top == 0)

{

temp->next = s->pop();

last = temp->next;

last->next = NULL;

}

}

}

void Print()

{

LinkListNode\* temp;

temp = start;

while (temp!= NULL)

{

cout << temp->data;

temp = temp->next;

}

}

};

int main()

{

Stack\* s = new Stack();

LinkList\* temp=new LinkList();

temp->AddNode(1);

temp->AddNode(2);

temp->AddNode(3);

temp->AddNode(4);

temp->AddNode(5);

cout << "Before Reversing" << endl;

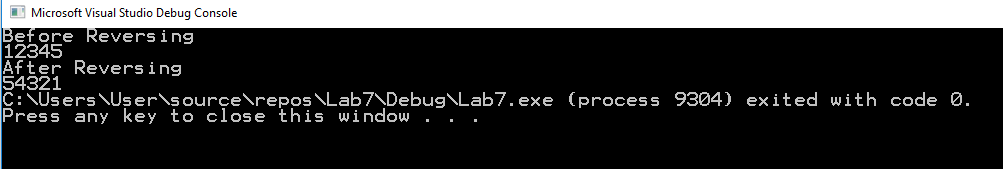
temp->Print();

cout << endl<< "After Reversing" << endl;

temp->Revesing();

temp->Print();

}



Loop for reversing the elements iterates n time so its time complexity for **Big-O is n**

**Task 8:**

In the below given table,

|  |  |  |
| --- | --- | --- |
| **Operation** | **Big-O Complexity** | **Best-case Complexity** |
| Insert an element at the front of a singly linked list of size n | 1 | 1 |
| Insert an element at the tail end of a singly linked list of size n. **plast** points to last node. | 1 | 1 |
| Delete the last node of a singly linked list of size n. **plast** points to its last node. | 1 | 1 |
| Insertion at the front of an array list of size n | 1 | 1 |
| Insertion at the tail end of an array list of size n | 1 | 1 |
| Enqueue in a queue of length n. | 1 | 1 |
| Dequeue in a queue of length n. | 1 | 1 |
| Converting an expression of length n from infix to postfix form using stack | n^2 | n^2 |
| Finding an element via Binary Search algorithm in a sorted array-list of size n. | Log2(n) | 1 |
| Finding an element via Binary Search algorithm in an **unsorted** array-list of size n. Think about it! | nLogn |  |

**Deliverable**

You are required to upload the lab tasks on LMS before the deadline.